The opinion in support of the decision being entered today was not written for publication and is not binding precedent of the Board.

Paper No. 17

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES

Ex parte GEOFFREY OWEN BLANDY

Appeal No. 2002-0273
Application No. 09/078,933

ON BRIEF

Before BARRETT, RUGGIERO, and BLANKENSHIP, Administrative Patent Judges.

BLANKENSHIP, Administrative Patent Judge.

DECISION ON APPEAL

This is a decision on appeal under 35 U.S.C. § 134 from the examiner's final rejection of claims 1-32, which are all the claims in the application.

We affirm-in-part.

BACKGROUND

The disclosed invention is directed to a method and apparatus for compiling instructions in a data processing system. In a JAVA implementation of the method, unexecuted paths within the method are not compiled or interpreted, but remain in bytecode form, minimizing storage utilization. Claim 1 is reproduced below.

> 1. A process in a data processing system executing a routine having a plurality of paths, wherein the routine has includes [sic] a plurality of first type instructions and wherein the data processing system executes second type instructions, the process comprising:
>
> identifying a path being executed, wherein the path is one of the plurality of paths in the routine and wherein a plurality of first type instructions are associated with the path; and
>
> translating the first type instructions for the path being executed, wherein first type instructions are translated into second type instructions for execution by the data processing system, wherein first type instructions for unexecuted paths within the routine remain untranslated.

The examiner relies on the following reference:

Kolawa et al. (Kolawa)	5,784,553	Jul. 21, 1998 (filed Apr. 30, 1997)

Claims 1-32 stand rejected under 35 U.S.C. § 102 as being anticipated by Kolawa.

We refer to the Final Rejection (Paper No. 9) and the Examiner's Answer (Paper No. 16) for a statement of the examiner's position and to the Brief (Paper No. 15) for appellant's position with respect to the claims which stand rejected.

OPINION

Kolawa

Kolawa is directed to a system for software testing. The reference describes an endeavor in creating an automated test suite generation tool for assisting a programmer in the generation and execution validation of test suites capable of covering most source code branches and fully exercising a program's functionality. Test suite generation involves creating a set of inputs which force the program to execute different parts of the source code. Col. 1, l. 26 - col. 2, l. 32.

Kolawa's system comprises, in general (Fig. 1), a test generation system (TGS) 10, which receives as an input a computer program expressed as source code 11. The TGS automatically generates as an output a test suite database 12.

The reference describes a JAVA implementation of the system at column 16, line 44 et seq. Kolawa teaches a JAVA platform 202 (Fig. 16), and compiling JAVA source code 200 into intermediate, portable bytecodes 210, which are not machine-specific. Col. 16, l. 62 - col. 17, l. 28. The bytecodes are a relatively high-level representation of the JAVA source code. Accordingly, in the context of Runtime System 222, the bytecodes are interpreted by JAVA interpreter 220, or turned into machine code by Just-In-Time Compiler 214. Col. 17, l. 51-65. The TGS provides capability to symbolically interpret a program in order to get symbolic information about the values of the program variables at any given point of execution, thus providing information on

how values of program variables depend on the inputs to the program. Col. 17, l. 66 - col. 18, l. 7.

The TGS may also be applied as a symbolic interpreter to find inputs that will cause a JAVA program to generate exceptions at runtime. The TGS Driver Program 224 reads the bytecodes, symbolically interprets them, and notifies the user of any possible runtime exceptions. Col. 18, ll. 22-31. The TGS Driver Program may also test the entire JAVA applet or application. Col. 18, ll. 56-67.

Kolawa describes the TGS Driver Program 224 in more detail at column 19, line 2 et seq. Initially, a random set of inputs are generated, and symbolic interpretation of the JAVA program is performed. The Driver Program includes a Symbolic Virtual Machine 234 (Fig. 17) for implementing symbolic interpretation, which in turn includes a Symbolic Execution Engine 332 (Fig. 19). The Symbolic Execution Engine 332 performs a function similar to an interpreter, but instead of assigning real values to each program variable, it assigns symbolic values to each program variable. The Execution Engine 332 symbolically executes the JAVA program under test. Col. 21, l. 66 - col. 22, li. 42.

Section 102 rejection of claims 1-32 over Kolawa

Appellant argues (Brief at 6-7) that Kolawa fails to disclose identifying a path being executed, where, as required by instant claim 1, the path is one of the plurality of paths in a routine. We disagree. The reference discloses symbolically interpreting and

executing a JAVA routine, expressed in bytecodes, in a sequential fashion. The path within the plurality of paths in the routine is necessarily identified; if not, then the path could not be interpreted and executed. Kolawa thus teaches identifying a path being executed, wherein the path is one of the plurality of paths in the routine, in the terms of instant claim 1.

Appellant also argues (Brief at 7-8) that Kolawa does not teach leaving unexecuted paths in bytecode form. Although the examiner has pointed to portions of the reference that disclose executing only selected paths, the language of instant claim 1 is not commensurate with appellant's argument. Even if Kolawa were to teach that all instructions in a program are to be executed, as appellant contends, the claim does not distinguish over the translation of bytecode instructions in a first of a plurality of paths, prior to translation of the remaining paths. According to our reading of Kolawa, the bytecode instructions in remaining paths remain untranslated at least until those remaining paths are translated and executed in the normal sequence of operation.

We are also unpersuaded by appellant's arguments in defense of claims 2 and 3, which appear to be based on the erroneous view that Kolawa fails to teach translating first type instructions (i.e., bytecode instructions) into another form. The "executing second type instructions" of claim 2 does not distinguish over the "symbolic" execution described by the reference, during normal looping within a program. Further, with respect to instant claim 3, Kolawa does not disclose that the translated instructions

are stored in anything other than execution order, and we find no reason to believe that the reference suggests otherwise.

While the "translating' recited by instant claim 1 is broad enough to read on the symbolic interpretation described by Kolawa, we note that the remainder of the independent claims are more specific. Claims 4 through 32 call for "compiling" the bytecodes.

Kolawa discloses compiling JAVA bytecodes by means of just-in-time Compiler 214 (Fig. 16). However, the compiling of the bytecodes is a separate operation from the symbolic interpretation and execution of the bytecodes effected by the TGS driver program 224. The just-in-time compilation described by the reference appears similar to that described at pages 4 and 5 of the instant specification, in the description of prior art methods.

The attempt by the rejection to read claims 4 through 32 on Kolawa appears to rely on just-in-time compiler 214 for the teaching of "compiling." However, compiler 214 is essentially unrelated to the further requirements of the claims. The rejection appears to, alternatively, rely on JAVA compiler 208 (Fig. 16) for the "compiling." Nonetheless, JAVA compiler 208, as made clear by Figure 16 and column 17, lines 20 through 28, is an intermediate compiler that compiles JAVA source code to JAVA bytecodes not specific to a particular machine, rather than compiling bytecodes into a form suitable for execution.

The reference refers to "interpreting" and "compiling" consistent with their customary meanings in the art. To "interpret" is understood to mean "[t]o analyze and execute each statement in a source program before translating and executing the next statement." IBM Dictionary of Computing at 355 (1994 ed.). The primary meaning of "compile" is to "translate all or part of a program expressed in a high-level language into a computer program expressed in an intermediate language, an assembly language, or a machine language." Id. at 125. "Compiling" thus connotes generating a computer program output, rather than step-by-step translation of statements.

In any event, the compilers to which the rejection refers are not part of the operation test generation systems or methods described by Kolawa, upon which the rejection further relies. The combinations set forth by claims 4-32 have thus not been shown in the reference. Cf. Lindemann Maschinenfabrik GmbH v. American Hoist & Derrick Co., 730 F.2d 1452, 1458, 221 USPQ 481, 485 (Fed. Cir. 1984) (anticipation requires the presence in a single prior art reference disclosure of each and every element of the claimed invention, arranged as in the claim).

For the foregoing reasons we sustain the rejection of claims 1-3, but do not sustain the rejection of claims 4-32.

CONCLUSION

The rejection of claims 1-3 under 35 U.S.C. § 102 is affirmed. The rejection of claims 4-32 under 35 U.S.C. § 102 is reversed. The examiner's decision is thus affirmed-in-part.

No time period for taking any subsequent action in connection with this appeal may be extended under 37 CFR § 1.136(a).

AFFIRMED-IN-PART

LEE E. BARRETT
Administrative Patent Judge)

JOSEPH F. RUGGIERO
Administrative Patent Judge) BOARD OF PATENT APPEALS AND INTERFERENCES

HOWARD B. BLANKENSHIP
Administrative Patent Judge)

Appeal No. 2002-0273
Application No. 09/078,933

DUKE E YEE
P O BOX 802334
DALLAS , TX 75380